Journal of Machine Learning Research 17 (2016) 1-5 Submitted 9/16; Revised 11/16; Published 12/16

RLScore: Regularized Least-Squares Learners

Tapio Pahikkala TAPIO.PAHIKKALA@UTU.FI
Antti Airola ANTTI.AIROLA@UTU.FI
Department of Information Technology
20014 University of Turku
Finland

Editor: Alexandre Gramfort

Abstract

RLScore is a Python open source module for kernel based machine learning. The library provides implementations of several regularized least-squares (RLS) type of learners. RLS methods for regression and classification, ranking, greedy feature selection, multi-task and zero-shot learning, and unsupervised classification are included. Matrix algebra based computational short-cuts are used to ensure efficiency of both training and cross-validation. A simple API and extensive tutorials allow for easy use of RLScore.

Keywords: cross-validation, feature selection, kernel methods, Kronecker product kernel, pair-input learning, python, regularized least-squares

1. Introduction

RLScore implements learning algorithms based on minimizing the regularized risk functional

$$\operatorname*{argmin}_{f \in \mathcal{H}} R(f) + \lambda \|f\|^2,$$

where f is the learned predictor, $\mathcal{H}$ a reproducing kernel Hilbert space of functions, $R(f)$ the empirical risk, $\|f\|^2$ the regularizer, and $\lambda > 0$ a regularization parameter.

Regularized least-squares[1] (RLS) is the classical method resulting from the choice $R(f) = \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2$. The method admits a closed form solution, leading to efficient algorithms for leave-one-out cross-validation (LOO), multi-target learning, and fast selection of regularization parameter (Rifkin and Lippert, 2007). For example, the LOO predictions can be obtained essentially for free as the sideproduct of computations needed for training the method once. Previously, these methods have been implemented in libraries such as GURLS (Tacchetti et al., 2013) and Python scikit-learn (Pedregosa et al., 2011).

In the recent years, research in RLS methods has lead to the development of a large variety of new efficient algorithms, that analogously to the classical RLS methods offer unique computational benefits both for training and model selection. These include leave-pair-out and leave-group-out cross-validation, methods for feature selection, ranking and unsupervised classification, as well as pair-input learning methods with applications to interaction prediction, cold start recommendations and zero-shot learning. RLScore is a Python module that provides a simple high-level interface to a library of highly optimized implementations of these methods.

1. aka kernel ridge regression, least-squares support vector machine

2. Implemented Algorithms

RLScore implements a large variety of fast holdout and CV algorithms. A fast leave-group-out (LGO) CV (Pahikkala et al., 2012b), where folds containing multiple instances are left out, is provided, complementing the classical fast RLS LOO algorithm (also included) (Rifkin and Lippert, 2007). The approach allows implementing fast K-fold CV, and more importantly, implementing CV for non i.i.d. data with natural group structure. Typical examples include leave-query-out CV for learning to rank, leave-sentence-out or leave-document out CV in text mining, leave-image-out CV for object recognition etc. Further, a leave-pair-out (LPO) algorithm (Pahikkala et al., 2009), that corresponds to leaving each combination of two instances (or a subset of these) from the data out in turn, is provided. LPO can be used to compute an almost unbiased estimate of area under ROC curve (Airola et al., 2011) and its generalization, the pairwise ranking accuracy.

RankRLS method implements efficient algorithms for both minimizing pairwise ranking losses and computing cross-validation estimates for ranking. The method has been shown to be highly competitive compared to ranking support vector machines (Pahikkala et al., 2009). Unsupervised variants of RLS classification inspired by the maximum margin clustering approach have also been developed (Pahikkala et al., 2012a).

Greedy RLS extends the basic RLS to learning sparse linear models in linear time, combining fast update formulas for feature addition and LOO with a greedy search (Naula et al., 2014). The computational short cuts allow scaling the approach to genome wide studies with hundreds of thousands of features. The method produced the winning submission of sub-challenge 3 of 2014 Broad-DREAM Gene Essentiality Prediction Challenge due to its ability to select a minimal accurate subset of features for multi-task learning problems.

RLS methods allow also fast learning from pair-input data. Applications include protein-protein and drug-target interaction prediction (Pahikkala et al., 2015), forecasting winners of two-player games, collaborative filtering, learning to rank for information retrieval etc. When making predictions for new pairs unseen in training set, the setting has natural applications in transfer and zero-shot learning. In the kernel methods framework this can be expressed as a learning problem where objects from two domains have their own kernel functions, and their joint kernel is the Kronecker product kernel. Efficient training and cross-validation algorithms for this setting have been recently derived (see e.g. Pahikkala et al. 2013; Stock et al. 2016).

3. Software Package

RLScore is implemented as a Python module that depends on NumPy (van der Walt et al., 2011) for basic data structures and linear algebra, SciPy (Jones et al., 2001–) for sparse matrices and optimization methods, and Cython (Behnel et al., 2011) for implementing low-level routines in C-language. The aim of the software is to provide high quality implementations of algorithms developed by the authors that combine efficient training with automated performance evaluation and model selection methods.

RLScore implements a modular design, where data representation, learning algorithms, and prediction are separated from each other where possible. The most basic kernel-based learning methods operate on a singular value decomposition of the data produced by an

adapter object. The hypothesis space used depends on the adapter, choices include both linear and kernel feature spaces, as well as Nyström type of reduced-set approximation. After training, the adapter creates a suitable type of linear or kernel predictor. The predictor object can be used or saved to disk independently of the algorithm used to train it.

The API design has been influenced by common Python data analysis environments such as NumPy, SciPy and scikit-learn, making it easy to combine RLScore with existing data analysis pipelines. The most fundamental classes in RLScore are learner objects in module `rlscore.learner`. At initialization, a learner is trained, and function `predict` is used for prediction. The predictor object can also be directly accessed and used independently of the learner. The majority of the learners also implement fast holdout and cross-validation functions, and support kernels, and fast multi-target learning. Unit tests are used to verify the implementations. Extensive tutorials describe how RLScore can be used to solve different types of problems. Listing 1 presents a simple demonstration of the interface.

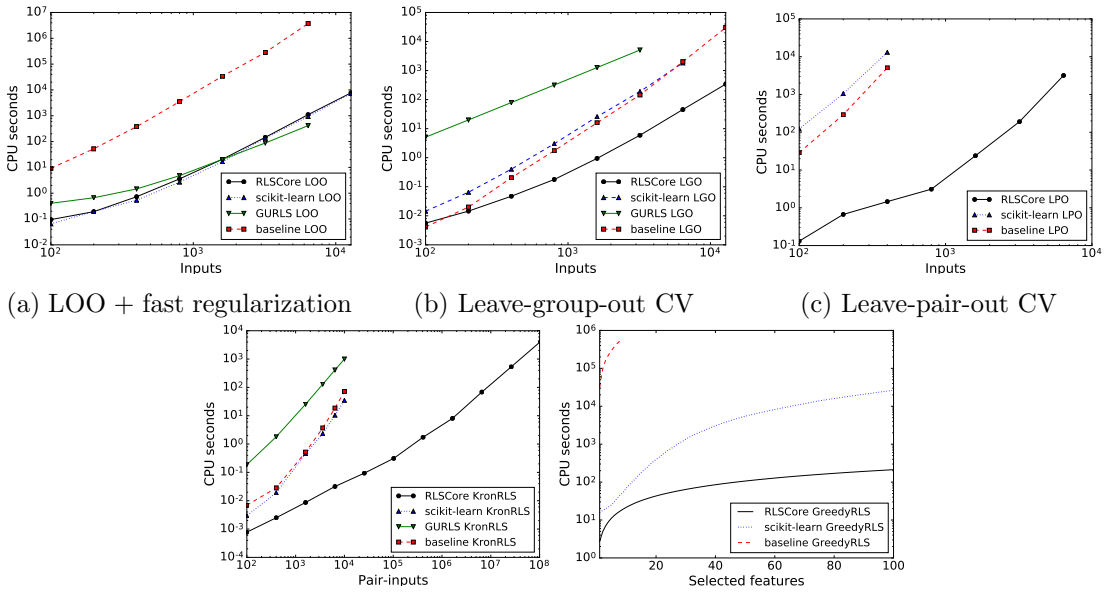
Listing 1: feature selection with greedy RLS algorithm

```
import numpy as np
from rlscore.learner import GreedyRLS
from scipy.stats import kendalltau

#regression problem with 3 important features
X = np.random.randn(100, 20)
y = X[:, 0] + X[:, 2] - X[:, 5] + 0.1*np.random.randn(100)
#select 3 features with greedy RLS
rls = GreedyRLS(X[:50], y[:50], regparam=1, subsetsize=3)
#Did we select the right features?
print(rls.selected)
#Compute test set predictions
p = rls.predict(X[50:])
print(kendalltau(y[50:], p))
```

4. Benchmarks

Here we demonstrate the advantages of RLScore solvers on five benchmark tasks. Each of the considered tasks can be expressed either as a single or a sequence of RLS problems with closed form solutions. The *baseline* method solves each resulting system $(\mathbf{K} + \lambda \mathbf{I})\mathbf{A} = \mathbf{Y}$ or $(\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})\mathbf{W} = \mathbf{X}^T\mathbf{Y}$ with Python `numpy.linalg.solve` that calls the LAPACK `_gesv` routine. Further we compare to two existing RLS solvers implemented in Python scikit-learn (version 0.18) (Pedregosa et al., 2011) and the MATLAB GURLS package (Tacchetti et al., 2013). The RLScore algorithms produce exactly the same results as the compared methods, but make use of a number of computational short-cuts resulting in substantial increases in efficiency. GURLS results were not included for LPO and feature selection as the runtimes were impractically long. Benchmark codes for comparing RLScore and scikit-learn RLS implementations are included in the RLScore code repository.



(a) LOO + fast regularization (b) Leave-group-out CV (c) Leave-pair-out CV

(d) Kronecker RLS (e) Feature selection

(a) Leave-one-out CV and regularization parameter selection (parameter grid $\{2^{-15}, ..., 2^{15}\}$, linear kernel, equal number of instances and features). Both scikit-learn and GURLS also implement fast LOO and regularization.

(b) Leave-group-out CV, 10 instances per fold, Gaussian kernel, 500 features.

(c) Leave-pair-out CV, Gaussian kernel, 500 features.

(d) Kronecker product kernel $\mathbf{K} \otimes \mathbf{G}$ is a popular choice in pair-input learning. `KronRLS` allows learning with the kernel without explicitly forming the pairwise kernel matrix. We generate two kernel matrices of size $n \times n$, the label vector $\mathbf{Y}$ contains n^2 entries, one label for each pair. Baseline explicitly constructs the $n^2 \times n^2$-sized kernel matrix.

(e) Learning sparse models. We consider greedy forward selection, where on each iteration one selects the feature whose addition provides the lowest RLS LOO error. `GreedyRLS` implements this procedure in linear time, with scikit-learn we use the fast LOO algorithm, baseline is a pure wrapper implementation. Data matrix $\mathbf{X}$ contains 10000 instances and 1000 features, and the number of outputs in $\mathbf{Y}$ is 10.

RLScore scales to orders of magnitude larger problem sizes than the baselines on all but the LOO experiment. With the exception of LOO, none of the considered fast algorithms are available in other software implementations. RLScore contains also a large variety of other methods, with new ones being added with each release.

Acknowledgments

We would like to acknowledge the support from the Academy of Finland (grants 134020 and 289903) and the co-authors who participated in developing the implemented algorithms.

References

A. Airola, Tapio Pahikkala, Willem Waegeman, Bernard De Baets, and Tapio Salakoski. An experimental comparison of cross-validation techniques for estimating the area under the ROC curve. *Computational Statistics & Data Analysis*, 55(4):1828–1844, 2011.

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39, 2011.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

P. Naula, A. Airola, T. Salakoski, and T. Pahikkala. Multi-label learning under feature extraction budgets. *Pattern Recognition Letters*, 40:56–65, 2014.

T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Järvinen, and J. Boberg. An efficient algorithm for learning to rank from preference graphs. *Machine Learning*, 75(1):129–165, 2009.

T. Pahikkala, A. Airola, F. Gieseke, and O. Kramer. Unsupervised multi-class regularized least-squares classification. In *IEEE International Conference on Data Mining*, pages 585–594. IEEE Computer Society, 2012a.

T. Pahikkala, H. Suominen, and J. Boberg. Efficient cross-validation for kernelized least-squares regression with sparse basis expansions. *Machine Learning*, 87(3):381–407, 2012b.

T. Pahikkala, A. Airola, M. Stock, B. De Baets, and W. Waegeman. Efficient regularized least-squares algorithms for conditional ranking on relational data. *Machine Learning*, 93(2–3):321–356, 2013.

Tapio Pahikkala, Antti Airola, Sami Pietilä, Sushil Shakyawar, Agnieszka Szwajda, Jing Tang, and Tero Aittokallio. Toward more realistic drug-target interaction predictions. *Briefings in Bioinformatics*, 16(2):325–337, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

R. Rifkin and R. Lippert. Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, Massachusetts Institute of Technology, Cambridge, USA, 2007.

Michiel Stock, Tapio Pahikkala, Antti Airola, Bernard De Baets, and Willem Waegeman. Efficient pairwise learning using kernel ridge regression: an exact two-step method. *CoRR*, abs/1606.04275, 2016.

A. Tacchetti, P. Mallapragada, M. Santoro, and L. Rosasco. GURLS: A least squares library for supervised learning. *Journal of Machine Learning Research*, 14(1):3201–3205, 2013.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, 2011.